The Parallel Pre-Processor: a Compiler for Distributed and Shared Memory Computers

Mark W. Govett[1]
National Oceanic and Atmospheric Administration
Forecast Systems Laboratory
Boulder, Colorado 80305, USA

Abstract

The goal in developing a robust parallelization tool is that it is easy to use, it requires minimal modifications to the original serial code, it is extensible to a wide variety of applications, and that it provide good portable performance. A directive-based parallelization tool is described called the Parallel Pre-processor (PPP) that meets most of these goals. The user inserts directives, in the form of comments, into existing Fortran code. PPP translates the code and directives into a parallel version that runs efficiently on shared and distributed memory high-performance computing platforms including: SGI Origin, IBM SP2, Cray T3E, SUN, Alpha and Intel Clusters. Twenty directives are available to support operations including array re-declarations, inter-process communications, loop transformations, and parallel I/O operations. PPP also provides support for incremental parallelization and parallel debugging.

Keywords: Directive-based parallelization, Weather and ocean models, parallel compiler, Fortran source to source translator, distributed memory computers

1. Introduction

Distributed memory computers (DMCs) are increasingly used to solve scientific problems because they offer superior price performance over their shared-memory counterparts. However, the biggest impediment toward the use of distributed memory computers is the lack of good high level programming models that run efficiently and are portable. Ideally these programming models should provide efficient execution of the application, access to simple high-level programming constructs, and be minimally invasive. Several programming models are supported by computer vendors and widely used by the scientific community. Currently, HPF, MPI and OpenMP are the most popular but none has proven effective at being sufficiently high level and offering scalable, portable performance.

High Performance Fortran (HPF) was developed to address the need to automatically parallelize Fortran codes. Despite support by parallel computer vendors and scientific institutions, it has largely failed to live up to the promise of high-performance and ease-of-use. The most basic short-coming of HPF is in the compiler's inability to determine the optimal location of inter-process communications efficiently. Code restructuring and the insertion of additional HPF directives can help the compiler do a better job, but this often requires additional effort by the programmer and performance is often significantly less than hand coded solutions. Additional research at Rice University has been done to create a distributed memory HPF (dHPF) to address the performance shortcomings and they have achieved some success [6]. However, performance still lags MPI-based hand coded solutions by at least 15 percent (often much more), and code restructuring is often required.

[1] E-mail address: govett@fsl.noaa.gov (M.Govett)

Since good performance is critical in most scientific applications, users have largely rejected HPF as a viable programming model. In order to achieve good results on DMCs, message passing libraries such as MPI are used. While the performance and scalability of parallel codes using message-passing libraries such as MPI can be quite good, they are relatively low-level and can require the programmer to expend a significant amount of effort to parallelize their code. Further, the resulting code may differ substantially from the original serial version and code restructuring is often desirable or necessary.

Another programming model commonly used to parallelize Fortran codes is OpenMP. OpenMP is a standard set of directives, designed in a collaboration between computer vendors and applications users, that has become widely used in the scientific community. OpenMP can be used to quickly produce parallel code, with minimal impact on the serial version. However, OpenMP is only available for shared memory computers. To get around this restriction, software developers have achieved limited success combining OpenMP (shared memory parallelism) with MPI (distributed memory parallelism) on hybrid DMCs. In either case however, obtaining good scalable performance often requires as much effort as when MPI is used directly.

These programming environments target Fortran codes since, despite its drawbacks, it remains the most popular programming language for scientific applications. In addition to these programming models, application specific frameworks have also emerged as a viable alternative to traditional language based approaches. In the atmospheric and ocean modeling community for example, the Earth System Modeling Framework (ESMF) [7] and the Program for Integrated Earth System Modeling [15] projects have begun recently with a goal of standardizing the interfaces between the framework and the applications that use them. Standard interfaces allow scientific codes to be more easily moved between models when they are built using the same framework. In addition to providing traditional library-based low-level communications routines, these approaches plan to support coupling of models and the robust handling of model grids that are required by climate applications. These tools are slated to be available in 2005, but at this time, it is unclear the extent to which source code changes will be required, or what the performance impacts will be in using them.

Despite the performance problems of HPF, and the limitations of OpenMP however, high-level directive-based parallelization remains a viable and attractive programming model for distributed memory. Directives offer a high level way to provide parallelization details to the compiler. Directives are lower level than modeling frameworks and thus permit flexibility and extensibility in handling wide ranging applications, yet are sufficiently high level as to allow common parallel operations to be encapsulated into simple high level directives.

Using this as a template to guide our development, we have built a compiler called the Parallel Pre-Processor (PPP) to translate Fortran source code and high level parallelization directives inserted by the programmer, into efficient parallel code. PPP directives provide similar functionality to HPF; however, in contrast to HPF where the placement of communications is determined by the compiler, PPP requires the programmer to determine the location of communications explicitly.

PPP has been used to parallelize many atmospheric and oceanic models including the Global Forecast System (GFS) [12] and the Typhoon Forecast System (TFS) [5] for the Central Weather Bureau in Taiwan, the Regional Ocean Modeling System (ROMS) [10], the Hybrid Coordinate Ocean Model (HYCOM) [2], the National Centers for Environmental Prediction (NCEP) Eta model [14], the high resolution limited area Quasi Non-hydrostatic model (QNH) [13], the Princeton Ocean Model (POM) [4], and the 20 km Rapid Update Cycle (RUC) model running operationally at NCEP [1]. These models run, without code change on a most HPCs including IBM SP2, Cray T3E, SGI Origin, Fujitsu VPP, and Alpha Linux clusters. Further, recent performance comparisons have demonstrated that the models parallelized using PPP runs as efficiently as their OpenMP and MPI counterparts.

The rest of this paper will describe PPP in more detail. Section 2 will describe PPP and highlight some advanced capabilities not available in other parallelization tools. Section 3 briefly describes two applications parallelized using PPP and compares their performance to OpenMP and MPI versions of these codes. Section 4 concludes and offers some insights and future directions of this approach.

2. Introduction to PPP

PPP is a Fortran code analysis and translation tool that was built using the Eli compiler construction system [9]. Eli is a domain specific programming environment that understands how to solve problems that are common to compiler development. In contrast to commonly used to tools such as LEX and YACC which were designed to generate scanners and parsers from simple specifications, Eli was designed to generate the whole compiler, given a high level specification. Specifications can be written that independently define the grammar, scanning, parsing, name and type analysis, symbol and definition table creation, language analysis, and the generation of the final output code. Eli will then do the analysis necessary, including determining a tree traversal strategy necessary to satisfy the specifications and generate a complete compiler. Eli builds the compiler either as a stand-alone executable or generates ANSI C code complete with makefile that can be ported to other systems.

PPP executables are used to analyze the serial code and user-inserted directives to determine how the code should be modified. Generated code relies on MPI based support libraries to perform operations including communications, parallel I/O, synchronization, local and global address space translation, and data layout. This library layer currently provides support for applications using regular structured grids that are solved using finite difference approximation (FDA) or spectral methods. Additionally, this library provides support for mesh refinement of nested models, and can transform data between grids that have been decomposed differently. Finally, this middle software layer provides a way in which to incorporate architectural-based optimizations, and make upgrades in message passing software transparent to the user. For example the upgrade of MPI from MPI-1 to MPI-2 requires only modifications to the communications layer and not to the application code. PPP and the run-time library, in combination, represent a parallelization tool called the Scalable Modeling System (SMS).

SMS has been designed to handle most aspects of parallel programming including data decomposition, reductions, handling of boundary conditions, static and dynamic memory, I/O, alignment, incremental parallelization, debugging and optimization operations. Access to parallization is primarily through twenty directives that are inserted into the application code. Some of these directives are similar to HPF. For example the DISTRIBUTE directive is used to identify the arrays that will be decomposed as it does in HPF$DISTRIBUTE. The DISTRIBUTE directive also supports the HPF$ALIGN capabililty via a keyword option. Many operations, such as I/O do not require any PPP directives; the translator simply determines how each array is decomposed and then generates the correct code.

However, the user is required to define the decomposition (DECLARE_DECOMP and CREATE_DECOMP). In addition, the type and placement of communications operations must be stated explicitly by the programmer. Three types of communication are currently supported: HALO_UPDATE (communicate halo region data between neighboring processors), TRANSFER (handle communication required between arrays that are decomposed differently), and REDUCE (a global reduction operation). To perform communication operations, the programmer simply specifies the arrays to be communicated and PPP will do the name and type analysis necessary to produce the correct translated code.

PPP does not currently do inter-procedural analysis so additional directives are required to determine if variables must be converted between local and global address spaces using TO_GLOBAL, TO_LOCAL and GLOBAL_INDEX. In addition, the PARALLEL directive is used to identify do-loops that require parallelization. While these directives are currently required, additional analysis by the compiler could eliminate the need for them. For example, PPP could determine parallel do-loop indices by identifying each decomposed array that is used within the loop and the indices used to reference them.

Further details about these directives and support for the given operations can be found a papers by Govett, et al [8]. PPP directives also provide support for incremental code parallelization and debugging support. This functionality is not provided by HPF or most parallelization tools and will be further described.

2.1 Incremental Parallelization

SMS provides support for simplifying code parallelization with a directive called SERIAL that permits serial execution of selected portions of the user's code. This directive has several important uses. First, it allows users to parallelize their code incrementally rather than being forced into an all-or-nothing approach. Once assured of correct results, the user can remove these serial regions and further parallelize their code. Second, this directive can be used to avoid the parallelization of some sections of code that are either executed infrequently (e.g. model initialization) or cannot be parallellized by PPP, such as NetCDF I/O [16].

Serial regions are implemented by gathering all decomposed arrays, executing the code segment on a single process, then scattering decomposed or broadcasting non-decomposed results back to each processor as illustrated in **Figure 1**. In this figure, the routine not_parallel executes on a single process and references global arrays that have been gathered by the appropriate SMS routines. While the extra communications required to do gather or scatter operations will slow performance, this directive's versatility has proven to be very useful during code parallelization.

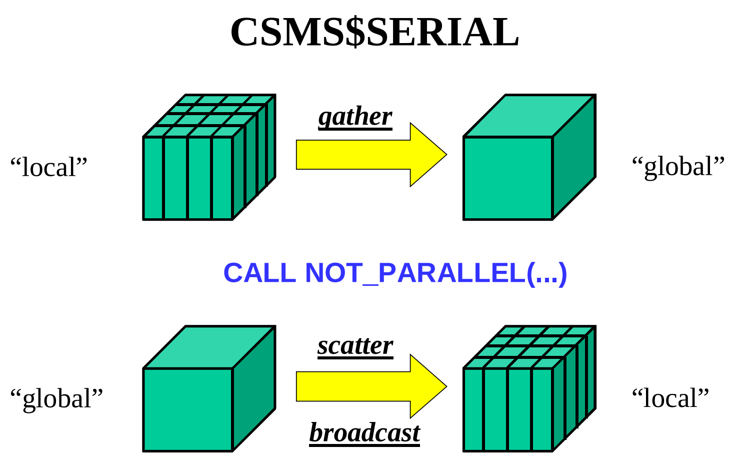

Figure 1: An illustration of how SMS supports incremental parallelization. Prior to execution of the serial region of code, decomposed arrays are gathered into global arrays, referenced by the serial section of code, and then results are scattered or broadcast back to the processors at the end of the serial region.

2.2 Debugging

Finding run-time bugs during the initial code parallelization or ensuing code maintenance phase can be the most difficult and time consuming task required in running codes on a DMC system. Two PPP directives have been developed to support debugging. As illustrated in **Figure 2**, the COMPARE_VAR directive is used to verify interior region data points are correct, and CHECK_HALO is used similarly for the halo points.

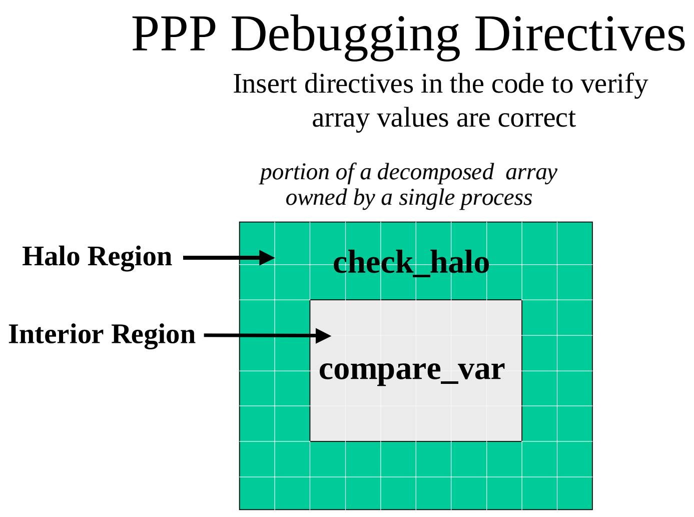

Figure 2: An illustration of two debugging directives that are available to verify decomposed array values are correct. Scalars and non-decomposed arrays can also be compared. These directives have greatly simplified debugging and parallel code development.

Using the CHECK_HALO directive, halo region values from each user-specified array are compared with their corresponding interior points on the neighboring process. When data values differ, SMS outputs an error message containing the array name, and the location where the problem occurred, and then terminates execution.

The COMPARE_VAR directive, patterned after work by O'Keefe [3], provides the ability to compare array or scalar values between a correctly working code and another run that uses different numbers of processors. For example, the programmer can specify a comparison of the array "x", for a single processor run and for a multiple process run by inserting the directive:

```
    csms$compare_var ( x )
```

in the code and then entering appropriate command line arguments to request concurrent execution of the code. Wherever COMPARE_VAR directives appear in the code, user-specified arrays will be compared as shown in **Figure 3**. If differences are detected, SMS will display the name of the variable, the array location (e.g., the i, j, k index) and values from each run, the location in the code, and then terminate execution. Conversely, if no differences are found, SMS will continue executing the code.

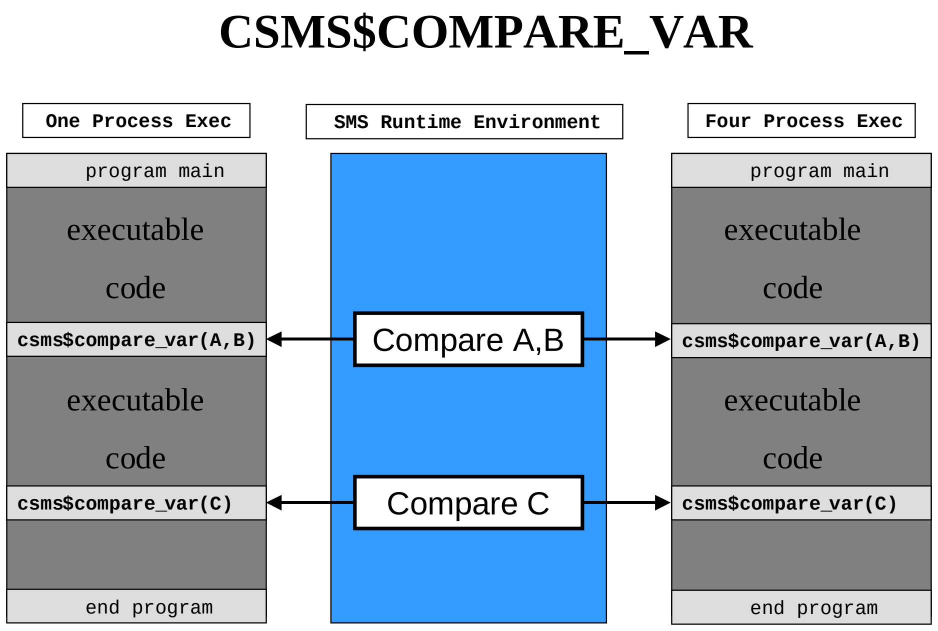

Figure 3: An illustration of how COMPARE_VAR is implemented. In this example, two executables are launched concurrently from the command line. When a COMPARE_VAR directive is encountered, the executables synchronize, and then compare the specified arrays. If any elements of the arrays differ, SMS will print the location and values of the data point and then terminate the execution of the runs.

The ability to compare intermediate model values anywhere in the code has proven to be a powerful debugging tool during code parallelization. For example, the time required to debug a recent code parallelization was reduced from an estimated eight weeks down to two simply because the programmer did not have to spend inordinate amounts of time determining where parallelization mistakes were made.

These directives have also proven to be a useful way to ensure that model upgrades continue to produce the correct results. For example, a scientist can verify source code changes by simply comparing the output files of the serial "control" run and subsequent parallel runs. In the event results differ, they can turn on SMS debugging (a run-time option) which compares the intermediate results of the arrays specified by COMPARE_VAR. In the event differences appear, they can quickly locate the problem and determine the best solution. In this way, SMS users have found the debugging directives very useful because they allow the code author to control the maintenance and upgrades of their parallel codes rather than requiring the help of a computer specialist.

3. Performance Comparisons

As a high-level software tool, SMS requires extra computations to maintain data structures that encapsulate low-level MPI functionality which could lead to potential performance degradation. To measure this impact, a performance comparison was done between the hand-coded MPI based version of the Eta model running operationally at NCEP, and the same Eta model parallelized using SMS. The MPI Eta model was considered a good candidate for fair comparison since it is an operational model used to produce daily weather forecasts for the U.S. National Weather Service and has been optimized for high performance on the IBM SP2. Fewer than 200 directives were added to the 19,000 line Eta model during SMS parallelization. Performance results show SMS-Eta was faster than MPI-Eta for all processor counts tested as illustrated in **Table 1**. Further analysis indicates that most of the performance gains in SMS-Eta were due to more efficient communications.

Number of Processors	MPI-Eta Time	SMS-Eta Time	SMS faster	SMS-Eta Efficiency
4	11197	10781	4 %	1.00
8	5317	5258	1 %	1.03
16	2878	2774	4 %	0.97
32	1471	1446	2 %	0.93
64	872	820	6 %	0.82
88	694	643	7 %	0.76

Table 1: Eta model performance for MPI-Eta and SMS-Eta run on NCEP's IBM SP-2. Run times are given in seconds for a full 48 hour model run including model initialization and the generation of hourly output files.

Another model studied in recent tests, compared the performance of a OpenMP version of the Regional Ocean Modeling System (ROMS), and an SMS parallelized version of the same code. The ROMS model is a ocean model developed jointly by UCLA and Rutgers University that is used by modeling groups around the world. One hundred-twenty directives were added to the 13,000 line ROMS code during SMS parallelization. This was equivalent to the number of OpenMP directives required. The SMS and OpenMP performance results, shown in **Table 2**, are for model runs on an SGI Origin 3000. Both one and two dimensional decompositions were used with the fastest run-times used in each case. In general, the fastest SMS run times were observed when data were decomposed in two dimensions, since communications scale better; whereas OpenMP did better when a one dimensional decomposition was used. In addition, the SMS one processor run was slower than the serial code because the SMS version used dynamic memory.

Number of Processors	OpenMP Time	SMS Time	SMS Efficiency	% SMS Faster
1	143.9	148.5	1.00	-3.1
2	70.0	74.4	1.00	-5.9
4	36.8	39.4	0.94	-6.5
8	19.6	20.7	0.90	-5.6
16	10.9	11.1	0.84	-1.8
32	6.9	6.5	0.71	5.8

Table 2: A comparison of OpenMP and SMS runs of the ROMS model on the SGI Origin 3000. The serial code ran in 144.7 seconds.

Further details of these performance studies are provided in a paper by Govett, et al. [8].

4. Conclusion

We have demonstrated that directive-based code parallelization is a viable method in which to parallelize codes for distributed memory computers. A compiler has been developed that provides simple high-level parallelization directives, requires no modifications to the original serial code, and demonstrates performance comparable to MPI or OpenMP. This tool has been used to parallelize a number of weather and ocean code for use on a variety of shared and distributed memory computers. The key to the success of this tool has been the code analysis and translation capabilities of PPP. This tool was built using an advanced compiler building tool called Eli that simplified the development tasks.

Further work remains to advance the analysis capabilities of PPP that will lead to fewer directives that must be inserted into the user code. Recent tests have shown that up to 50 percent of the directives can be eliminated by more extensive code analysis. Parallelization using PPP could become largely determining how and which arrays are decomposed, and the placement of communications in the serial code.

We believe the success of this work demonstrates the usefulness of the approach. While this tool has only been applied to weather and ocean models the approach is sufficiently general to be applied to other application areas. We would like this development to serve as a prototype for a general purpose compiler that can bridge the gap between low-level yet high performance message passing libraries and high-level but sub-optimal performance programming models such as HPF.

References

[1] S.Benjamin, J.Brown , K.Brundage, D.Dévényi, G.Grell, D.Kim, B.Schwartz, T.Smirnova, T.Smith, S.Weygandt and G.Manikin, RUC 20 – The 20-km version of the Rapid Update Cycle, National Weather Service Technical Procedures Bulletin No. 490 (2002), http://ruc.fsl.noaa.gov/ppp_pres/RUC20-tpb.pdf .

[2] R.Bleck, An Oceanic General Circulation Model Framed in Hybrid Isopycnic-Cartesian Coordinates. Submitted to J. Ocean Modeling *(2001)*.

[3] R.Bleck, S.Dean, M.O'Keefe and A.Sawdey, A comparison of data-parallel and message passing versions of the Miami Isopycnic Coordinate Ocean Model (MICOM), Parallel Computing, 21 (1995).

[4] A.F.Blumberg and G. L. Mellor, A description of a three-dimensional coastal ocean circulation model, Three-Dimensional Coastal ocean Models, edited by N. Heaps, 208 pp., American Geophysical Union (1987).

[5] D.S.Chen, K.N.Huang, T.C.Yeh, M.S.Peng , and S.W.Chang, Recent improvements of the typhoon forecast system in Taiwan. 23th Conference on Hurricanes and Tropical Meterology. Dallas, TX., (2000) 823-825.

[6] The dHPF Compiler Project. http://www.cs.rice.edu/~dsystem/dhpf/overview.html.

[7] Earth System Modeling Framework Development Team, http://www.esmf.ucar.edu/.

[8] M.Govett, L.Hart, T.Henderson, J.Middlecoff, and D.Schaffer, The Scalable Modeling System: Directive-Based Code Parallelization for Distributed and Shared Memory Computers. Submitted to Journal of Parallel Computing (2002).

[9] R.Gray, V.Heuring, S.Levi, A.Sloane, and W.Waite, Eli, A Flexible Compiler Construction System, Communications of the ACM 35 (1992) 121-131.

[10] D.B.Haidvogel, H.G.Arango, K.Hedstrom, A.Beckman, P.Malanotte-Rizzoli, and A.F Shchepetkin, Model Evaluation Experiments in the North Atlantic Basin: Simulations in Nonlinear Terrain-Following Coordinates, Dyn. Atmos. Oceans 32 (2000) 239-281.

[11] T.Henderson, C.Baillie, S.Benjamin, T.Black, R.Bleck, G.Carr, L.Hart, M.Govett, A.Marroquin, J.Middlecoff and B.Rodriguez, Progress Toward Demonstrating Operational Capability of Massively Parallel Processors at Forecast Systems Laboratory, Proceedings of the Sixth *ECMWF Workshop on the Use of Parallel Processors in Meteorology*, *European Centre for Medium Range Weather Forecasts*, Reading, England (1994).

[12] C.S.Liou, J.Chen, C.Terng, F.Wang, C.Fong, T.Rosmond, H.Kuo, C.Shiao, and M. Cheng, The Second-Generation Global Forecast System at the Central Weather Bureau in Taiwan, Weather and Forecasting 12 (1997) 653-663.

[13] A.E.MacDonald, J.L.Lee, and Y.Xie, QNH: Design and Test of a Quasi Non-hydrostatic Model for Mesoscale Weather Prediction. Monthly Weather Review 128 (2000) 1016-1036.

[14] F.Mesinger, The Eta Regional Model and its Performance at the U.S. National Centers for Environmental Prediction. International Workshop on Limited-area and Variable Resolution Models. Beijing, China, WMO/TD 699 (1995) 42-51.

[15] Program for Integrated Earth System Modeling (PRISM) Web Page: http://prism.hnes.org/.

[16] R.K.Rew and G.P.Davis, Unidata's netCDF Interface for Scientific Data Access, Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology, Anaheim, CA (1990).